RFC: HDF5 File Space Management: Paged Aggregation

Vailin Choi

Quincey Koziol

John Mainzer

The current HDF5 file space allocation accumulates small pieces of metadata and raw data in aggregator blocks, which are not page aligned and vary widely in sizes. To provide efficient paged access to these small pieces of metadata and raw data, we propose a file space management mechanism we call Paged Aggregation.

# Table of Contents:

Table of Contents: 2

1 Introduction 4

2 Overview of Current File Space Management 4

2.1 Public Routines 5

2.2 File Space Info Message 6

3 Overview of Paged Aggregation 7

4 Public routines 8

4.1 Additions to the API 8

4.2 Modifications to the API 9

5 File Space Info message 9

6 Cycle of operation when allocating file space 11

6.1 H5F\_FSPACE\_STRATEGY\_FSM\_AGGR 11

6.2 H5F\_FSPACE\_STRATEGY\_PAGE 11

6.3 H5F\_FSPACE\_STRATEGY\_AGGR 12

6.4 H5F\_FSPACE\_STRATEGY\_NONE 12

7 Cycle of operation when freeing file space 12

7.1 H5F\_FSPACE\_STRATEGY\_FSM\_AGGR 12

7.2 H5F\_FSPACE\_STRATEGY\_PAGE 13

7.3 H5F\_FSPACE\_STRATEGY\_AGGR 14

7.4 H5F\_FSPACE\_STRATEGY\_NONE 14

8 Cycle of operation when shrinking file space 14

8.1 H5F\_FSPACE\_STRATEGY\_FSM\_AGGR 15

8.2 H5F\_FSPACE\_STRATEGY\_PAGE 16

8.3 H5F\_FSPACE\_STRATEGY\_AGGR 16

8.4 H5F\_FSPACE\_STRATEGY\_NONE 16

9 Cycle of operation when extending file space 16

9.1 H5F\_FSPACE\_STRATEGY\_FSM\_AGGR 16

9.1.1 Extending the section at EOA 16

9.1.2 Extending the section into either the metadata or raw data aggregator 16

9.1.3 Extending the section into a free-space section 17

9.2 H5F\_FSPACE\_STRATEGY\_PAGE 17

9.2.1 Extending the section at EOA 17

9.2.2 Extending the section into a free-space section 17

9.2.3 Extending the section into the page-end threshold (may delete this optimization) 17

9.3 H5F\_FSPACE\_STRATEGY\_AGGR 18

9.3.1 Extending the section at EOA 18

9.3.2 Extending the section into either the metadata or raw data aggregator 18

9.4 H5F\_FSPACE\_STRATEGY\_NONE 18

9.4.1 Extending the section at EOA 18

10 Shutting Down Free-Space Managers on File Close 18

10.1 H5F\_FSPACE\_STRATEGY\_FSM\_AGGR 19

10.2 H5F\_FSPACE\_STRATEGY\_PAGE 20

10.3 H5F\_FSPACE\_STRATEGY\_AGGR 20

10.4 H5F\_FSPACE\_STRATEGY\_NONE 20

11 Floating Free-Space Managers When Reopening Files 21

11.1 H5F\_FSPACE\_STRATEGY\_FSM\_AGGR 21

11.2 H5F\_FSPACE\_STRATEGY\_PAGE 21

11.3 H5F\_FSPACE\_STRATEGY\_AGGR 21

11.4 H5F\_FSPACE\_STRATEGY\_NONE 21

12 Tools 22

12.1 h5dump 22

12.2 h5stat 22

12.3 h5repack 22

13 Testing 22

14 Documentation 23

15 Backward/Forward Compatibility 24

16 Limitations 24

17 A New Solution For Shutting Down Free-space Managers 24

17.1 H5FS\_sinfo\_unlock() 25

17.2 Cache Callbacks For Free-space Manager 26

17.3 Shutting Down Free-Space Managers on File Close 27

17.3.1 H5F\_FSPACE\_STRATEGY\_FSM\_AGGR and H5F\_FSPACE\_STRATEGY\_PAGE 27

17.3.2 H5F\_FSPACE\_STRATEGY\_AGGR and H5F\_FSPACE\_STRATEGY\_NONE 28

17.4 Allocation Of Cache Image At File Close 28

17.5 Clean up 28

17.6 Backward and Forward Compatibility 28

17.7 Testing 29

17.8 Time estimate 29

18 Future Issues 29

Acknowledgements 29

Revision History 29

# Introduction

This document addresses the changes to the HDF5 library needed to implement the file space handling mechanism we call paged aggregation. This mechanism aggregates small metadata and raw data allocations into constant-sized well-aligned pages, which are suitable for page caching. Paged aggregation together with the *Page Buffering* feature should allow more efficient I/O accesses.

The detailed rationale behind this proposed mechanism is described in the *RFC: HDF5 File Space Allocation and Aggregation*. The *Page Buffering* feature is described in the *RFC: Page Buffering*.

The new features discussed in this RFC have been implemented and released in HDF5 1.10.1. Until such time as a document discussing file free space management in HDF5 is written, this document should be updated to reflect any API or algorithmic changes in the features discussed.

# Overview of Current File Space Management

At present, the HDF5 library uses three mechanisms to manage space in an HDF5 file. They are:

* Free-space managers

They track sections of the file of various sizes that are currently free or unallocated. Each free-space manager corresponds to a file space type. There are two main groups of file space types: metadata and raw data. Metadata is further divided into five types: superblock, B-tree, global heap, local heap, and object header.

* Aggregators

The library manages two aggregators, one for metadata and one for raw data (but note that metadata aggregation is turned off In the split and multi file drivers). The aggregator is a contiguous block of free space in the file. The size of each aggregator is tunable via public routines *H5Pset\_meta\_block\_size* and *H5Pset\_small\_data\_block\_size* respectively.

The current implementation of the aggregator blocks is not page-aligned, and the sizes may vary from the original specified block size.

* Virtual file drivers

The library’s virtual file driver interface dispatches requests for additional space to the allocation routine of the current file driver, which manages I/O to the file. For example, if the *sec2* file driver is being used, its allocation routine will increase the size of the file to service requests.

For files with contiguous address space, the default behavior is to have one free-space manager to handle metadata and one free-space manager to handle raw data.

For files with non-contiguous address space, it is possible to have up to one free-space manager for each of the six file space types: raw data and five types of metadata.

Based on these mechanisms, there are four file space handling strategies available to users for managing file space:

1. H5F\_FILE\_SPACE\_ALL

* Mechanisms used: free-space managers, aggregators, and virtual file drivers
* Free space managed by the free-space managers is discarded at file close, and is thus not persistent across file close/open cycles.
* This strategy is the library default

1. H5F\_FILE\_SPACE\_ALL\_PERSIST

* Mechanisms used: free-space managers, aggregators, and virtual file drivers
* Free space managed by the free-space managers is saved at file close, and thus persists across file close/open cycles

1. H5F\_FILE\_SPACE\_AGGR\_VFD

* Mechanisms used: aggregators and virtual file drivers
* Free-space is not retained across file close/open cycles

1. H5F\_FILE\_SPACE\_VFD

* Mechanisms used: virtual file drivers
* Free-space is not retained across file close/open cycles

Please refer to the *HDF5 File Space Management* document for full details.

## Public Routines

* *herr\_t H5Pset\_file\_space(hid\_t fcpl, H5F\_file\_space\_type\_t strategy, hsize\_t threshold)*
* Set the file space handling *strategy* and free-space section *threshold* in the file creation property list *fcpl*; the setting cannot be changed for the life of the file.
* *strategy* is the file space handling strategy defined as:

typedef enum H5F\_file\_space\_type\_t {

H5F\_FILE\_SPACE\_DEFAULT=0, /\* Not Used?? \*/

H5F\_FILE\_SPACE\_ALL\_PERSIST=1, /\* Persistent FSM, aggregators, VFD \*/

H5F\_FILE\_SPACE\_ALL=2, /\* Non-persistent FSM, aggregators, VFD \*/

H5F\_FILE\_SPACE\_AGGR\_VFD=3, /\* Aggregators, VFD \*/

H5F\_FILE\_SPACE\_VFD=4, /\* VFD \*/

H5F\_FILE\_SPACE\_NTYPES

} H5F\_file\_space\_type\_t;

* + *threshold* is the smallest free-space section size that the free-space manager will track.
* *herr\_t H5Pget\_file space(hid\_t fcpl, H5F\_fspace\_strategy\_t \*strategy, hsize\_t \*threshold)*
  + Retrieve the file space handling strategy and free-space threshold value in the parameters *strategy* and *threshold* respectively.
  + Return the library default value as follows when not set via *H5Pset\_file\_space*:
    - *strategy—* H5F\_FILE\_SPACE\_ALL
    - *threshold—*1

## File Space Info Message

The library’s default setting for handling file space is:

* File space handling strategy is H5F\_FILE\_SPACE\_ALL
* Free-space section threshold is 1

If the user sets file space info that deviates from any of the above, the library will create the *File Space Info* message to store the non-default setting and the message is stored in the superblock extension.

**Layout: Version 0 File Space Info message**

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte** | **Byte** | **Byte** | **Byte** |
| Version | File space strategy | *This exists to align table nicely.* | |
| Free-space section thresholdL | | | |
| AddressesO of free-space managers for the six file space types:  H5FD\_MEM\_SUPER, H5FD\_MEM\_BTREE, H5FD\_MEM\_DRAW,  H5FD\_MEM\_GHEAP, H5FD\_MEM\_LHEAP, H5FD\_MEM\_OHDR | | | |

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| Version | The version number is used to indicate the format of the message. The value is 0. |
| File space strategy | This is the file space strategy used to manage file space:   * H5F\_FILE\_SPACE\_ALL * H5F\_FILE\_SPACE\_ALL\_PERSIST * H5F\_FILE\_SPACE\_AGGR\_VFD * H5F\_FILE\_SPACE\_VFD |
| Free-space section threshold | The smallest free-space section size that the free-space manager will track. |
| Addresses of free-space managers | The addresses of free-space managers for the six file space types when strategy is H5F\_FILE\_SPACE\_ALL\_PERSIST. |

# Overview of Paged Aggregation

The goal of paged aggregation is to accumulate metadata and raw data into well-aligned pages, which we call file space pages. The library defines a default file space page size but user can set the page size via a new public routine, *H5Pset\_file\_space\_page\_size*.

The free-space manager mechanism is modified to handle paged aggregation as follows:

* Small-sized free-space manager:
  + Track free-space sections whose size is < file space page size
  + Satisfy requests either from existing free space if available; if not, request a page from large-sized manager:
    - Returned space: no page alignment constraint; cannot cross page boundary
  + Shrink a free-space section:
    - Never – but release sections of size equal to the page size to the large-sized free-space manager, which may do so.
  + Merge two free-space sections:
    - When the two sections adjoin and they are on the same page
    - When the merged section is equal to page size, return to the large-sized free-space manager
* Large-sized free-space manager
  + Track free-space sections whose size is >= file space page size, along with smaller, misaligned sections left over from allocations >= file space page size.
  + Satisfy requests either from existing free space if available; if not, request space from virtual file driver
    - Returned space: page aligned; can cross page boundary
  + Shrink a free-space section:
    - When the section ends at EOA and the section is >= page size
    - To keep EOA at page boundary: shrink only full-sized pages but retain partial pages in the manager
  + Merge two free-space sections when they adjoin

For files with contiguous address space, the default behavior is to have two small-sized free-space managers (one for metadata and one for raw data) and one large-sized free-space manager for generic data (can be metadata or raw data).

For files with non-contiguous address space, it is possible to have up to 6 small-sized free-space managers and 6 large-sized free-space managers. They correspond to the six file space types: raw data and five types of metadata.

We propose four file space-handling strategies available to users for handling file space:

1. H5F\_FSPACE\_STRATEGY\_FSM\_AGGR:
   * Mechanisms used: free-space managers, aggregators, and virtual file drivers
   * This is the library default, and is the same as the current library default
2. H5F\_FSPACE\_STRATEGY\_PAGE:

* Mechanisms used: free-space managers with embedded paged aggregation and virtual file drivers

1. H5F\_FSPACE\_STRATEGY\_AGGR:

* Mechanisms used: aggregators and virtual file drivers

1. H5F\_FSPACE\_STRATEGY\_NONE:

* Mechanisms used: virtual file driver

For all the above strategies, the default is to not retain free-space across file close/open cycles. The user can use the public routine *H5Pset\_file\_space\_strategy()* to request persistent file free-space, which is retained across file close/open cycles. Requests for persistent file free space management are not applicable to the last two strategies, as they do not use file free space managers.

# Public routines

## Additions to the API

* *herr\_t H5Pset\_file\_space\_page\_size(hid\_t fcpl, hsize\_t fsp\_size)*
* Set the file space page size *fsp\_size* for paged aggregation in the file creation property list *fcpl*; the size set via this routine cannot be changed for the life of the file.
* *fsp\_size* has a minimum size of 512. Setting value less than 512 will return an error.
* The library default value for file space page size when not set is 4096.
* The maximum page size is 1 GB. Attempts to set page size larger than this value will result in an error.
* *herr\_t H5Pget\_file\_space\_page\_size(hid\_t fcpl, hsize\_t \*fsp\_size)*
* Retrieve the file space page size for paged aggregation in the parameter *fsp\_size* from the file creation property list *fcpl*.
* Return the library default 4KB (4096) in *fsp\_size* If the page size is not set via *H5Pset\_file\_space\_page\_size*.
* *herr\_t H5Pset\_file\_space\_strategy(hid\_t fcpl, H5F\_fspace\_strategy\_t strategy, hbool\_t persist, hsize\_t threshold)*
  + Set the file space handling *strategy*, whether free space should *persist across file close/open cycles,* and free-space section *threshold* in the file creation property list *fcpl*; the setting cannot be changed for the life of the file.
* *strategy* is the file space handling strategy defined as:

typedef enum H5F\_fspace\_strategy\_t {

H5F\_FSPACE\_STRATEGY\_FSM\_AGGR = 0, /\* FSM, Aggregators, VFD \*/

H5F\_FSPACE\_STRATEGY\_PAGE = 1 /\* Paged FSM, VFD \*/

H5F\_FSPACE\_STRATEGY\_AGGR = 2 /\* Aggregators, VFD \*/

H5F\_FSPACE\_STRATEGY\_NONE = 3, /\* VFD \*/

H5F\_FSPACE\_STRATEGY\_NTYPES

} H5F\_fspace\_strategy\_t;

* + *persist* indicates whether file free space should be retained across file close/open cycles.
  + *threshold* is the smallest free-space section size that the free-space manager will track.
  + As H5F\_FSPACE\_STRATEGY\_AGGR and H5F\_FSPACE\_STRATEGY\_NONE strategies do not use free-space managers, the *persist* and *threshold* settings will be ignored for those strategies.
* *herr\_t H5Pget\_file space strategy(hid\_t fcpl, H5F\_fspace\_strategy\_t \*strategy, hbool\_t \*persist, hsize\_t \*threshold)*
  + Retrieve the file space handling strategy, free-space persistence, and threshold value in the parameters *strategy*, *persist* and *threshold* respectively.
  + Return the library default value as follows when not set via *H5Pset\_file\_space\_strategy*:
    - *strategy—* H5F\_FSPACE\_STRATEGY\_FSM\_AGGR
    - *persist—*FALSE
    - *threshold—*1

## Modifications to the API

* *herr\_t H5Pset\_alignment(hid\_t plist, hsize\_t threshold, hsize\_t alignment)*
* Add the description to the reference manual entry that if H5F\_FSPACE\_STRATEGY\_PAGE strategy is used, the alignment set via this routine is ignored.

# File Space Info message

The library’s default setting for handling file space is:

* File space strategy is H5F\_FSPACE\_STRATEGY\_FSM\_AGGR
* Not persisting free-space
* Free-space threshold is 1
* File space page size is 4096

If the user sets file space info that deviates from any of the above, the library will create the *File Space Info* message to store the non-default setting and the message is stored in the superblock extension.

**Layout: Version 1 File Space Info message**

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte** | **Byte** | **Byte** | **Byte** |
| Version | File space strategy | Persisting free-space | *This exists to align table nicely.* |
| Free-space section thresholdL | | | |
| File space page sizeL | | | |
| Page-end metadata threshold | | *This exists to align table nicely.* | |
| EOAO | | | |
| AddressesO of small-sized free-space managers for the six file space types:  H5FD\_MEM\_SUPER, H5FD\_MEM\_BTREE, H5FD\_MEM\_DRAW,  H5FD\_MEM\_GHEAP, H5FD\_MEM\_LHEAP, H5FD\_MEM\_OHDR | | | |
| AddressesO of large-sized free-space managers for the six file space types:  H5FD\_MEM\_SUPER, H5FD\_MEM\_BTREE, H5FD\_MEM\_DRAW,  H5FD\_MEM\_GHEAP, H5FD\_MEM\_LHEAP, H5FD\_MEM\_OHDR | | | |

LThis itemin thetable is the *size of lengths* as defined in the superblock.

OThis item in thetable is the *size of offsets* as defined in the superblock.

**Fields: File Space Info Message**

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| Version | The version number is used to indicate the format of the message. The value is 1. |
| File space strategy | This is the file space strategy used to manage file space:   * H5F\_FSPACE\_STRATEGY\_FSM\_AGGR * H5F\_FSPACE\_STRATEGY\_PAGE * H5F\_FSPACE\_STRATEGY\_AGGR * H5F\_FSPACE\_STRATEGY\_NONE |
| Persisting free-space | True or False in persisting free-space |
| Free-space section threshold | The smallest free-space section size that the free-space manager will track.  This optimization is currently disabled, and may be deleted from the code. |
| File space page size | The file space page size, which is used when paged aggregation is enabled. |
| Page-end metadata threshold | The smallest free-space section size at the end of a page that the free-space manager will track. This is used when paged aggregation is enabled. |
| EOA | The EOA before the allocation of free-space header and section info for the self-referential\* free-space managers when the library is persisting free-space. |
| Addresses of free-space managers | The addresses of small-sized free-space managers for the six file space types when persisting free-space. |
| Addresses of free-space managers | The addresses of large-sized free-space managers for the six file space types when persisting free-space and when paged aggregation strategy is enabled. |

\*Please see description of self-referential free-space managers in ***Shutting down free-space managers on file close*** (see section 10 below).

# Cycle of operation when allocating file space

The library calls the routine *H5M\_alloc()* to request file space when creating objects for the HDF5 file. The mechanisms used to fulfill the request will depend on the file space strategy with the cycle of operation described below.

## H5F\_FSPACE\_STRATEGY\_FSM\_AGGR

* The library will request space from the free-space manager depending on the file space type.
* If the request can be satisfied by the appropriate free space manager, the library will do so and return the address to the caller.
* If the request cannot be satisfied by the appropriate free space manager, the library will request space from either the metadata or raw data aggregator depending on the file space type.
* If the request can be satisfied by the appropriate aggregator, the library will do so and return the address to the caller.
* If the request cannot be satisfied by the appropriate aggregator, the library will request space from the virtual file driver and return the address to the caller.

## H5F\_FSPACE\_STRATEGY\_PAGE

* The library will request space from the free-space manager depending on the request size:
  + For request size < page size, request is sent to the small-sized free-space manager for the requested file space type.
  + For request size >= page size, request is sent to the large-sized free-space manager for the requested file space type.
* If the request can be satisfied from existing free space, the free space manager will do so, and return the address to the caller.
* If the request cannot be satisfied from existing free space:
  + The small-sized free space manager will request a page of file space from the large-sized free-space manager for the file space type, satisfy the file space request from the newly allocated page, and return the address of the new space to the caller.
  + The large-size free space manager will request sufficient pages from the virtual file driver, satisfy the request out of the newly allocated space, and return the address of the new space to the caller.

Since EOA must always be on a page boundary, the large-sized free space manager will retain any mis-aligned left over space so that it can be merged with the allocation that caused it when and if that allocation is released.

## H5F\_FSPACE\_STRATEGY\_AGGR

* The library will request space from either the metadata or raw data aggregator depending on the file space type.
* If the request can be satisified by the appropriate aggregator, the library will do so and return the address to the caller.
* If the request cannot be satisfied by the appropriate aggregator, the library will request space from the virtual file driver and return the address to the caller.

## H5F\_FSPACE\_STRATEGY\_NONE

* The library will request space from the virtual file driver and return the address to the caller.

# Cycle of operation when freeing file space

The library calls the routine *H5MF\_xfree()* when releasing file space with the cycle of operation described below.

## H5F\_FSPACE\_STRATEGY\_FSM\_AGGR

* If there is no existing free-space manager for the file space type, the library will attempt shrinking action as described in ***Cycle of operation when shrinking file space*** (see section 8 below).
* If the shrinking action succeeds, return to the caller.
* If the section size is less than the free-space threshold, drop the section on the floor and return to the caller.
* If the section size is >= free-space threshold, start up the free-space manager for the file space type and perform the following:
  + Try merging the section with existing sections in the free-space manager if they adjoin.
  + Try shrinking action as described in ***Cycle of operation when shrinking file space*** (see section 8 below).
  + If the section is not merged away or shrunk, add the section to the manager.
  + Return to the caller.

## H5F\_FSPACE\_STRATEGY\_PAGE

* If there is no existing free-space manager for the file space type, the library will attempt shrinking action as described in ***Cycle of operation when shrinking file space*** (see section 8 below).
* If the shrinking action succeeds, return to the caller.
* If the section size is less than the free-space threshold, drop the section and return to the caller.
* If the section size is >= free-space threshold, start up the small-sized or large-sized free-space manager for the file space type and perform the following (Note: we are experimenting with deleting the page-end threshold region optimization, as we question whether the added complexity is worth the gain.):
  + For a small-sized section:
    - If the section resides in the page-end threshold region, drop the section and return to the caller.
    - If the section ends within the page-end threshold region, increase section size to end of page and continue.
  + Try merging the section with existing sections:
    - For a small section: if they adjoin and the merged section does not cross page boundary.
    - For a large section: if they adjoin.
* Try shrinking action as described in ***Cycle of operation when shrinking file space*** (see section 8 below)
* If the section is not merged away or shrunk, add the section to the corresponding manager.
* Return to the caller.

## H5F\_FSPACE\_STRATEGY\_AGGR

* The library will attempt shrinking action as described in ***Cycle of operation when shrinking file space*** (see section 8 below)***.***
* If the shrinking action succeeds, return to the caller.
* Otherwise drop the section on the floor and return to the caller.

## H5F\_FSPACE\_STRATEGY\_NONE

* The library will attempt shrinking action as described in ***Cycle of operation when shrinking file space*** (see section 8 below)***.***
* If the shrinking action succeeds, return to the caller.
* Otherwise drop the section on the floor and return to the caller.

# Cycle of operation when shrinking file space

The library performs shrinking actions via the *can\_shrink()* and *shrink()* callbacks according to the free-space section class. There are 3 kinds of section classes:

* *simple* section class:
  + Used by the following strategies:
    - H5F\_FSPACE\_STRATEGY\_FSM\_AGGR
    - H5F\_FSPACE\_STRATEGY\_AGGR
    - H5F\_FSPACE\_STRATEGY\_NONE
  + Attempt 2 kinds of shrinking action:
    - Shrink via EOA: shrink the file at EOA by section size
    - Shrink via aggregator:
      * Try merging the section into the aggregator
      * Try absorbing the aggregator into the section
      * Applicable only for H5F\_FSPACE\_STRATEGY\_FSM\_AGGR and H5F\_FSPACE\_STRATEGY\_AGGR strategies
* *small* section class:
  + The *can\_shrink*() and *shrink*() callbacks are not defined for the small section class. However, the small free space managers will release complete pages to the appropriate large free space manager when they become free.
* *large* section class:
  + Used by the H5F\_FSPACE\_STRATEGY\_PAGE strategy for the large-sized manager.
  + Attempt 1 kind of shrinking action:
    - Shrink via EOA: shrink the file at EOA when the section size is >= file space page size.

The *can\_shrink()* callback determines the shrinking action that can be done and the *shrink()* callback actually performs the shrinking action allowed. The following data structure is used to pass information to/from the client and the callbacks:

typedef struct H5MF\_sect\_ud\_t {

/\* Down \*/

H5F\_t \*f; /\* Pointer to file to operate on \*/

hid\_t dxpl\_id; /\* DXPL for VFD operations \*/

H5FD\_mem\_t alloc\_type; /\* File space type \*/

hbool\_t allow\_sect\_absorb; /\* See below \*/

hbool\_t allow\_eoa\_shrink\_only; /\* See below \*/

/\* Up \*/

H5MF\_shrink\_type\_t shrink; /\* See below \*/

H5F\_blk\_aggr\_t \*aggr; /\* Aggregator block to operate on \*/

} H5MF\_sect\_ud\_t;

* *allow\_sect\_absorb:* whether the section is allowed to absorb the aggregator
  + TRUE: allow the section to absorb the aggregator
  + FALSE: does not allow the section to absorb the aggregator i.e. only allow merging the section into the aggregator; this setting is mainly used when starting up the free-space manager is not desirable
* *allow\_eoa\_shrink\_only:* whether only *shrink via EOA* is allowed
  + TRUE: allow only *shrink via EOA*; this setting is mainly used when shutting down the free-space managers on file closing
  + FALSE: no restriction
* *shrink:* type of shrink operation to perform as determined by *can\_shrink()* callback
  + **H5MF\_SHRINK\_EOA: section should shrink the EOA value**
  + **H5MF\_SHRINK\_AGGR\_ABSORB\_SECT: section should merge into the aggregator**
  + **H5MF\_SHRINK\_SECT\_ABSORB\_AGGR: aggregator should merge into the section**

## H5F\_FSPACE\_STRATEGY\_FSM\_AGGR

* If the section ends at EOA, shrink the file by section size and return to the caller.
* Otherwise, try *shrink via aggregator*:
  + If the section adjoins the beginning or end of the aggregator, merge the section into the aggregator or absorb the aggregator into the section.

## H5F\_FSPACE\_STRATEGY\_PAGE

* For a small section at EOA or elsewhere, if the section is page aligned, and the section size is equal to file space page size, release the section to the large-size free-space manager. Note that this is part of the normal operation of the small free space managers, and is not triggered via the *can\_shrink*() and *shrink*() callbacks.
* For a large section at EOA, shrink the file if the section size is >= file space page size. Note that only full-sized pages are shrunk with partial page put into the large-sized manager to keep EOA at page boundary.

## H5F\_FSPACE\_STRATEGY\_AGGR

* If the section ends at EOA, shrink the file by section size and return to the caller.
* Otherwise, try *shrink via aggregator*:
  + If the section adjoins the beginning or end of the aggregator, merge the section into the aggregator.

## H5F\_FSPACE\_STRATEGY\_NONE

* If the section ends at EOA, shrink the file by section size and return to the caller.

# Cycle of operation when extending file space

Under appropriate circumstances, the library will attempt to extend an existing section of allocated memory via the *H5MF\_try\_extend()* call. The cycle of operation for this action is described below.

## H5F\_FSPACE\_STRATEGY\_FSM\_AGGR

### Extending the section at EOA

* If the section ends at EOA:
  + Extend the file by extra requested and return *extended* to the caller.
* Otherwise continue with 9.1.2.

### Extending the section into either the metadata or raw data aggregator

* If the section adjoins the beginning of the aggregator:
* If the aggregator is not at EOA:
* If the aggregator has enough space to fulfill the extra requested:
  + Extend the section into the aggregator and return *extended* to the caller.
* Otherwise continue with 9.1.3.
* If the aggregator is at EOA:
* If the extra requested is below the extension percentage threshold:
  + Extend the section into the aggregator and return *extended* to the caller.
* If the extra requested is above the extension percentage threshold:
* Increase the size of the aggregator, extend the section into the aggregator and return *extended* to the caller.
* Otherwise continue with 9.1.3

### Extending the section into a free-space section

* If the section adjoins an existing free-space section in the manager with size large enough to fulfill the extra requested:
  + Extend the section into the adjoined free-space section and return *extended* to the caller.
* Otherwise return *not extended* to the caller.

## H5F\_FSPACE\_STRATEGY\_PAGE

### Extending the section at EOA

* If the section ends at EOA:
  + For a small-sized section:
    - Return *not extended* to the caller.
  + For a large-sized block:
    - Extend the file by extra requested plus misaligned fragment to keep the EOA at page boundary; put the misaligned fragment into the large-sized free-space manager.
  + Return *extended* to the caller.
* Otherwise continue with 9.2.2.

### Extending the section into a free-space section

* If the section adjoins an existing free-space section in the manager with size large enough to fulfill the extra requested:
  + Extend the section into the adjoined free-space section and return *extended* to the caller.
* Otherwise continue with 9.2.3.

### Extending the section into the page-end threshold (may delete this optimization)

* For a metadata section which ends in the page-end threshold region and the threshold size can fulfill the extra requested:
  + Extend into the threshold region and return *extended* to the caller.
* Otherwise return *not extended* to the caller.

## H5F\_FSPACE\_STRATEGY\_AGGR

### Extending the section at EOA

* If the section ends at EOA:
  + Extend the file by extra requested and return *extended* to the caller.
* Otherwise continue with 9.3.2.

### Extending the section into either the metadata or raw data aggregator

* If the section adjoins the beginning of the aggregator:
* If the aggregator is not at EOA:
* If the aggregator has enough space to fulfill the extra requested:
  + Extend the section into the aggregator and return *extended* to the caller.
* Otherwise return *not extended* to the caller.
* If the aggregator is at EOA:
* If the extra requested is below the extension percentage threshold:
  + Extend the section into the aggregator and return *extended* to the caller.
* If the extra requested is above the extension percentage threshold:
* Increase the size of the aggregator, extend the section into the aggregator and return *extended* to the caller.
* Otherwise return *not extended* to the caller.

## H5F\_FSPACE\_STRATEGY\_NONE

### Extending the section at EOA

* If the section ends at EOA:
* Extend the file by extra requested and return *extended* to the caller.
* Otherwise return *not extended* to the caller.

# Shutting Down Free-Space Managers on File Close

When the file is closing down and persistent free-space management is enabled, the library will allocate space for all raw data and then call *H5MF\_settle\_raw\_data\_fsm()* to allocate file space for the free-space managers that are not self-referential, It will then call *H5MF\_settle\_meta\_data\_fsm()* to allocate file space for the self-referential free-space managers. These two settle routines are called before the final close of file space via *H5MF\_close().*

Self-referential free-space managers are managers that allocate file space for free space managers’ free-space header and/or section info. The reasons to differentiate these managers from others are:

* The allocation of file space for header and section info may result in the managers being empty, thus forcing us to free the space.
* The allocation of file space for section info may change the size of the section info itself, thus forcing us to free the space.

To avoid the above potential problems that may result in infinite loop, the routine *H5MF\_settle\_raw\_data\_fsm()* will first settle file space for managers that are not self-referential by:

* Reduce the file’s EOA to the extent possible
* Allocate file space for the *file space info* message
* Allocate file space for the header and section info for free space managers that are not self-referential

After calling this routine, all raw data allocations are finalized as well as metadata allocations not involving self-referential free space managers.

Then the routine *H5MF\_settle\_meta\_data\_fsm()* will settle file space by:

* Reduce the file’s EOA to the extent possible
* Allocate space at the end of the file for the header and section info of the self referential free space managers directly from the VFD so as to avoid changing the contents of the free space managers.

As allocation of space for the self referential free space managers directly from the virtual file driver could increase the file size on each subsequent file close/open cycle, the self referential free space manager are floated on file open, and the file space allocated to them is freed – reducing the EOA to its value prior to the allocation of space for the self referential free space managers. (see section 11, ***Floating free-space managers when reopen file*** below for further details).

Note: *H5FD\_free()* and *H5FD\_alloc()* mentioned below are the virtual file driver’s *free* and *alloc* callbacks to free and allocate file space.

## H5F\_FSPACE\_STRATEGY\_FSM\_AGGR

The *H5MF\_settle\_raw\_data\_fsm()* will perform the following:

* Free file space for the metadata and raw data aggregators via *H5MF\_xfree()*, if any.
* Free file space (if any) allocated for each free-space manager’s header and section info via *H5MF\_xfree()*.
* Delete the *file space info* message from the superblock extension if allocated.
* Shrink the file’s EOA to the extent possible via *H5FD\_free().*
* Re-allocate file space for the *file space info* message.
* For each non empty and non self referential free space manager, re-allocate file space for the free-space manger’s header and section info via *H5MF\_alloc()*

The *H5MF\_settle\_meta\_data\_fsm()* will perform the following:

* Free file space for the metadata and raw data aggregators via *H5MF\_xfree()*.
* Shrink the file’s EOA to the extent possible via *H5FD\_free()*.
* Save the EOA in *f->shared->eoa\_pre\_fsm\_fsalloc* which is the EOA before the allocation of header and section info for self-referential managers.
* For each non empty, self referential free space manager, allocate file space from EOA via *H5FD\_alloc()* for each free-space manager’s header and section info.

## H5F\_FSPACE\_STRATEGY\_PAGE

The *H5MF\_settle\_raw\_data\_fsm()* will perform the following:

* Free file space allocated for each free-space manager’s header and section info via *H5MF\_xfree()*; this is done for the small-sized and large-sized free-space managers.
* Delete the *file space info* message from the superblock extension if allocated.
* Shrink the file’s EOA to the extent possible via *H5FD\_free()*.
* Re-allocate file space for the *file space info* message.
* For each non empty and non self referential free space manager, re-allocate file space for each free-space manger’s header and section info via *H5MF\_alloc()*.

The *H5MF\_settle\_meta\_data\_fsm()* will perform the following:

* Shrink the file’s EOA to the extent possible via *H5FD\_free()*.
* Save the EOA in *f->shared->eoa\_pre\_fsm\_fsalloc* which is the EOA before the allocation of header and section info for self-referential managers.
* For each non empty, self referential free space manager, allocate file space at EOA via *H5FD\_alloc()* for each free-space manager’s header and section info. The file space allocated is extended to the next file space page boundary.

## H5F\_FSPACE\_STRATEGY\_AGGR

Since no free-space managers are involved with this strategy, the library will just perform final close of file space via *H5MF\_close()*.

## H5F\_FSPACE\_STRATEGY\_NONE

Since no free-space managers are involved with this strategy, the library will just perform final close of file space via *H5MF\_close()*.

# Floating Free-Space Managers When Reopening Files

When the file is re-opened with read/write access and persisting free-space, the library will float the self-referential free-space managers and free the file space allocated to them so as to avoid ever increasing file space for each subsequent file close/open.

To do this, the library calls *H5MF\_tidy\_self\_referential\_fsm\_hack()* on the first file space allocation or de-allocation or other operation that references the free space managers. The activities of this function are described below.

## H5F\_FSPACE\_STRATEGY\_FSM\_AGGR

* Get the current EOA. If the EOA is the same as *f->shared->eoa\_pre\_fsm\_fsalloc*, we are done and return to the caller.
* Load and open any self referential free space managers whose header and section info appears after the address stored in *f->shared->eoa\_pre\_fsm\_fsalloc*.
* Float the above self referential free space managers via *H5FS\_free(),* which will remove the free space manager header and section info from the metadata cache.
* Free the file space allocated to the self referential free space manager header and section info via *H5FD\_free()*. The EOA after this operation should be *f->shared->eoa\_pre\_fsm\_fsalloc.*

## H5F\_FSPACE\_STRATEGY\_PAGE

* Get the current EOA. If the EOA is the same as *f->shared->eoa\_pre\_fsm\_fsalloc*, we are done and return to the caller.
* Load and open any self referential small-size free space managers whose header and section info appears after the address stored in *f->shared->eoa\_pre\_fsm\_fsalloc*.
* Load and open any self referential large-size free space managers whose header and section info appears after the address stored in *f->shared->eoa\_pre\_fsm\_fsalloc*.
* Float the above self referential free space managers via *H5FS\_free()* which will remove the free space manager header and section info from the metadata cache.
* Free the file space allocated to the self referential free space manager header and section info via *H5DF\_free()*. The EOA after this operation should be *f->shared->eoa\_pre\_fsm\_fsalloc.*

## H5F\_FSPACE\_STRATEGY\_AGGR

Since no free-space managers are involved for this strategy, there is no need to float the managers on the first file space allocation or de-allocation when reopening the file.

## H5F\_FSPACE\_STRATEGY\_NONE

Since no free-space managers are involved for this strategy, there is no need to float the managers on the first file space allocation or de-allocation when reopening the file.

# Tools

## h5dump

When printing the file creation property information for superblock via the -B option, include the page size obtained via *H5Pget\_file\_space\_page\_size*.

## h5stat

When printing file space information via the -S option, include the page size obtained via *H5Pget\_file\_space\_page\_size*.

## h5repack

* Add a new option *-G* FS\_PAGEISZE
  + Set the file space page size to FS\_PAGESIZE via *H5Pset\_file\_space\_page\_size*
* Add a new option -P FS\_PERSIST
  + Set persisting free-space to FS\_PERSIST via *H5Pset\_file\_space\_strategy*
  + FS\_PERSIST can be 1 (persist) or 0 (not persist)
  + Options already implemented to set file space info via *H5Pset\_file\_space\_strategy*:
    - -S FS\_STRATEGY to set the file space management strategy
    - -T FS\_THRESHOLD to set free-space section threshold

# Testing

Two files, *mf.c* and *tfile.c*, in the *test* directory contain tests to verify the correctness of paged aggregation:

Tests in *mf.c*:

* *test\_page\_small()*
  + verify that small-sized allocations and de-allocations of file space function as described for paged aggregation.
* *test\_page\_large():* 
  + verify that large-sized allocations and de-allocations of file space function as described for paged aggregation.
* *test\_page\_small\_try\_extend()*
  + verify that the extension of an allocated block that is < file space page size function as described for paged aggregation*.*
* *test\_page\_large\_try\_extend()*
  + verify that the extension of an allocated block that is >= file space page size function is as described for paged aggregation.
* *test\_page\_try\_shrink()*
  + verify that shrinking an allocated block that is large or small sized will function as described for paged aggregation.
* *test\_page\_alloc\_xfree()*
  + verify for both persisting or non-persisting free-space, the allocations and de-allocations of file space function as described for paged aggregation*.*
* *test\_page\_alignment()*
  + verify that alignment set via H5Pset\_alignment() or paged aggregation strategy work as described for page aggregation.

Tests in *tfile.c*:

* *test\_userblock\_alignment\_paged()*
  + verify that the public routines *H5Pset\_userblock*, *H5Pset\_alignment* and paged aggregation work as expected.
* *test\_filespace\_info()*
  + verify that the public routines *H5Pget/set\_file\_space strategy* and *H5Pget/set\_file\_space\_page\_size* work as expected.
* *test\_file\_freespace()*
  + verify that the public routine *H5Fget\_freespace()* returned the expected amount of free-space.
* *test\_sects\_freespace()*
  + verify that the public routine *H5Fget\_free\_sections()* returned the expected information for free-space sections.

The tests in *test/dsets.c, test/stab.c,* and *test/fheap.c* are modified to run with and without paged aggregation enabled.

# Documentation

Add reference manual entries for the following two new public routines:

* *H5Pset/get\_file\_space\_page\_size*
* *H5Pset/get\_file\_space\_strategy*

Update the following reference manual entries about the changes described previously due to paged aggregation:

* *H5Pset\_alignment*
* Tools: *h5repack*, *h5dump*, *h5stat*
* *Version 1 File Space Info* message in *HDF5 File Format Specification*
* Modify description in *HDF5 File Space Management* to reflect file space handling strategies due to paged aggregation

# Backward/Forward Compatibility

The first compatibility issue concerns HDF5 1.10.0, which was released with the file free space management as described in section 2. For release 1.10.1, paged aggregation as described in this RFC will be used instead. In the following description, we refer to the file space management in release 1.10.0 as the old method and that in release 1.10.1 as the new method. The compatibility issue will be addressed as below:

* The public routines *H5Pset/get\_file\_space()* will be deprecated in release 1.10.1.
* When opening a file that is created under release 1.10.0 with the old method, the library will read the old *File Space Info* message with version 0 and map the information to the new *File Space Info* message with version 1. Then the library will delete the old message and create the new message with the mapped information in the superblock extension. This means the file will have the version 1 *File Space Info* message from this point on.
* Generate HDF5 files with the old method under library release 1.10.0. File a bug report about the format change and deposit the files there for future reference.

Another compatibility issue is the possibility of a round trip scenario between library release 1.10.1 that supports the new method and library releases like 1.8 that do not understand the new method:

* Create an HDF5 file and dataset under library release 1.10.1
* Open the HDF5 file and modify the dataset under library release 1.8
* Open the HDF5 file and modify the dataset again under library release 1.10.1

This issue is addressed by enabling the H5O\_MSG\_FLAG\_MARK\_IF\_UNKNOWN flag in the *File Space Info* message when a file is created under library release 1.10.1. When the 1.8 library opens the file, it will mark the *File Space Info* message as unknown by setting the H5O\_MSG\_FLAG\_WAS\_UNKNOWN flag. Later on, when the 1.10.1 library opens the file again, it will detect the flag being set and will operate with the library default file space handling.

# Limitations

Currently, when multi/split driver is used in combination with paged aggregation strategy and/or persistent free-space, file creation/open will fail.

# A New Solution For Shutting Down Free-space Managers

Section 10 and section 11 describes a temporary fix to avoid the potential infinite loop problem when allocating file space for the meta data of the self-referential free-space managers at file closing. In this section, we propose a solution to properly settle this infinite loop issue by not allowing the allocation of section info to shrink in size at file close. This means when allocating file space for the section info with a specified size *x*, if the section info shrinks in size during the process, the initial size *x* will be retained and the section info allocated may be larger than what is actually needed. This fix will settle the allocation of meta data for self-referential free-space managers within a few passes.

The following five areas are identified for the implementation of the new solution:

* *H5FS\_sinfo\_unlock()*
  + Fix to disallow the shrinking of section info
* The cache *serialize* and *deserialize* callbacks for the free-space manager
  + Fix to accommodate unused space in section list on disk
* Shutting down free-space managers on file close
  + Fix the *H5MF\_settle\_meta\_data\_fsm()* routine to allocate file space for header and section info via *H5MF\_alloc()*
* Allocation of cache image at file close
  + Fix to allocate the cache image block via *H5MF\_alloc()*
* Clean up the existing code

## H5FS\_sinfo\_unlock()

The handling of a free-space section for the free-space manager is enclosed with the *H5FS\_sinfo\_lock()/H5FS\_sinfo\_unlock()* pair of routines. The actions that might be performed on a free-space section are:

* Adding a section via *H5FS\_sect\_add*()
* Removing a section via *H5FS\_sect\_remove*()
* Locating a section that is large enough to fulfill a space request via *H5FS\_sect\_find*()
* Extending a section into an existing section via *H5FS\_sect\_try\_extend*()
* Merging a section with existing sections via *H5FS\_sect\_try\_merge*()
* Shrinking an existing section if it is at EOF via *H5FS\_sect\_try\_shrink\_eoa*()
* Iterating over all the free-space sections via *H5FS\_sect\_iterate*
* Changing the class of a section via *H5FS\_sect\_change\_class*()

The *H5FS\_sinfo\_lock()* routine ensures the section info for the free-space manager is in memory as follows:

* If the free-space manager already owns the section info:
  + Ensures that a protected section info has read/write access
* If the free-space manager does not own the section info, performs either one of the following actions:
  + Load the section info from disk if it exists and protects it
  + Create and initialize a section info

The *H5FS\_sinfo\_unlock()* performs the following to release the section info if it is changed. The routine will be modified to only allow expansion of section info on file close:

* If the input parameter indicates that the section info has been modified:
  + Mark the section info for the free-space manager as dirty and modified
  + Mark the header for the free-space manager as dirty
* After decrementing, if the lock count for the section info goes to zero:
  + If the section info is protected:
    - If the section info has not been modified, unprotect the section info via *H5AC\_unprotect()*
    - If the section info has been modified, unprotect the section info as dirty. It might also be unprotected with flushing to disk and marked to be released later if either one of the following conditions is true:
      * The file is not closing down and the section info size has changed
      * The file is closing down and the section info size has increased

If the file is closing down but the section info size has decreased, the section info size is set to the previous allocated section size.

* + If the section info is not protected:
    - If the section info exists and has been modified, mark the section info to be released later if either one of the following conditions is true:
      * The file is not closing down
      * The file is closing down and the section info size has increased

If the file is closing down but the section info size has decreased, the section info size is set to the previous allocated section size.

* If the section info is marked to be released, free the section info via *H5MF\_xfree()*

## Cache Callbacks For Free-space Manager

Modify the *serialize* and *deserialize* callbacks for the free-space manager as follows:

* Modify the *deserialize* callback for section info to decode the free-space section list with possible unused space after the checksum
* Modify sanity checks to allow unused space after the checksum
* Modify sanity checks to allow for zero section counts

See the *HDF5 Format Specification* for the format of the free-space section list on disk.

## Shutting Down Free-Space Managers on File Close

We will follow the same flow of control when the file is closing down and persistent free-space management is enabled. The library will first allocate file space for the free-space managers that are not self-referential via *H5MF\_settle\_raw\_data\_fsm()* which is unchanged. It will then call *H5MF\_settle\_meta\_data\_fsm()* which will be modified as described below to allocate file space for the self-referential free-space managers.

### H5F\_FSPACE\_STRATEGY\_FSM\_AGGR and H5F\_FSPACE\_STRATEGY\_PAGE

For these two strategies with persisting free-space, *H5MF\_settle\_raw\_data\_fsm()* will perform the same actions as before:

* Free file space for the metadata and raw data aggregators via *H5MF\_xfree()*, if any.
* Free file space (if any) allocated for each free-space manager’s header and section info via *H5MF\_xfree()*.
* Delete the *file space info* message from the superblock extension if allocated.
* Shrink the file’s EOA to the extent possible via *H5FD\_free().*
* Re-allocate file space for the *file space info* message.
* For each non empty and non self referential free space manager, re-allocate file space for the free-space manger’s header and section info via *H5MF\_alloc()*

The *H5MF\_settle\_meta\_data\_fsm()* will be modified as follows:

* Free file space for the metadata and raw data aggregators via *H5MF\_xfree()*.
* Shrink the file’s EOA to the extent possible via *H5FD\_free()*.
* For each non empty, self referential free space manager, allocate file space via *H5MF\_alloc()* for each free-space manager’s header and section info. Repeat this action until all the header and section info for the self referential free-space managers are allocated. This should be settled in couple passes.
* After allocating the file space for all the free-space managers, get the current EOA and set it to *f->shared-> eoa\_pre\_fsm\_fsalloc*:
  + This will allow the 1.10 library with the hack to open the file with the new fix. The actions in *H5MF\_tidy\_self\_referential\_fsm\_hack()* will be bypassedas follows*:*
    - It will first obtain the EOA
    - If the EOA is the same as *f->shared->eoa\_pre\_fsm\_fsalloc*, it will exit the routine with SUCCESS

For the H5F\_FSPACE\_STRATEGY\_FSM\_AGGR strategy, when *H5MF\_alloc()* is unable to find file space from the free-space managers, it will call *H5MF\_\_aggr\_alloc()* to get space from the aggregators and the virtual file drivers. This routine will be modified to skip allocation from the aggregators when the file is closing down to avoid starting up the aggregators again.

### H5F\_FSPACE\_STRATEGY\_AGGR and H5F\_FSPACE\_STRATEGY\_NONE

Since no free-space managers are involved, nothing is changed for these two strategies. The library will just perform final close of file space via *H5MF\_close()*.

## Allocation Of Cache Image At File Close

When the file is closing down, the library will call *H5AC\_prep\_for\_file\_close()* to notify the metadata cache that the file is about to be closed. This routine will invoke *H5C\_\_prep\_image\_for\_file\_close()* to create a metadata cache image if this has been requested. Instead of allocating the cache image block via *H5FD\_alloc()* so as to avoid unsettling the free-space managers, we will now use *H5MF\_alloc()*. This will also take care of freeing a potential fragment resulted from alignment other than 1 instead of dropping the fragment to the floor.

## Clean up

* *f->shared->first\_alloc\_dealloc*
  + Remove this field which is used to keep track of persistent free space managers that have not been accessed for allocation or de-allocation since file open
  + Remove coding that set and check for *f->shared->first\_alloc\_dealloc*
  + Remove the associated routine *H5F\_get\_first\_alloc\_dealloc()*
* *H5MF\_tidy\_self\_referential fsm\_hack()*
  + Remove this routine since we no longer need to float free-space managers when reopening files
  + Remove coding that invokes *H5MF\_tidy\_self\_referential\_fsm\_hack()* based on *f->shared->first\_alloc\_dealloc* in the following routines:
    - *H5MF\_alloc()*
    - *H5MF\_xfree()*
    - *H5MF\_try\_extend()*
    - *H5MF\_try\_close()*
    - *H5MF\_get\_free\_sections()*
* *H5C\_force\_cache\_image\_load()*
  + Remove this routine since we no longer use *H5MF\_tidy\_self\_referential\_fsm\_hack()*
  + Remove coding that invokes this routine

## Backward and Forward Compatibility

To enable the 1.10 library with the hack to read a file with the new fix, the *f->shared->eoa\_pre\_fsm\_fsalloc* field is set to the EOA after settling the meta data for the self-referential free-space managers. See the description in section 17.3.1.

When the library with the new fix opens a file generated by the 1.10 library with the hack, the library is not doing anything to float the self-referential free-space managers nor free the file space allocated to them. For the H5F\_FSPACE\_STRATEGY\_PAGE strategy in particular, this means the unused file space allocated at EOA for the self-referential free-space managers’ header and section info will not be reclaimed and is dropped to the floor.

## Testing

* Ensure *test/cache\_image.c* succeeds
* Ensure *test/fheap.c* succeeds when running with HDF5TestExpress set to 0
* Add new tests to *test/mf.c* to verify the following cases:
  + The free-space manager goes empty when allocating space for the free-space header
  + Section info decreases, increases or stays the same
  + Alignment is not 1 resulting in fragment via *H5FD\_alloc()*
* Compatibility testing:
  + Create test file from 1.10 library with the hack and ensure that the current library with the new fix can access the file
  + Create test file from the current library with the fix and ensure the 1.10 library with the hack can access the file

## Time estimate

* Estimate 50 hours for the implementation and debugging of this new solution
* Estimate 60 hours for testing as described in the previous section
* Estimate 20 hours for RFC and code review

# Future Issues

TBD

# Acknowledgements

This work was supported by Lawrence Livermore National Laboratory (LLNL). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author[s] and do not necessarily reflect the views of LLNL.

# Revision History

|  |  |
| --- | --- |
| *August 22, 2012* | Version 0 – Modifications based on *RFC: HDF5 File Space Allocation and Aggregation* by John Mainzer |
| *January 10, 2016* | Version 1 –Updated to reflect current implementation. |
| *January 13, 2016* | Version 2—Updated after review. |
| *June 26, 2017* | Version 3 – Updated to clean up prose, and to bring the document into conformance with the current state of the code. |
| *April 19, 2018* | Version 4—Accepted changes from John Mainzer and checked in. |
| *June 19, 2018* | Version 5—Document the new solution for shutting down free-space managers on file close |